Presented at the *1994 Symposium on Low-Power Electronics*, San Diego, CA, October 1994

# Compilation Techniques for Low Energy: An Overview

Vivek Tiwari Sharad Malik Andrew Wolfe

Dept. of Electrical Engineering
Princeton University, Princeton, NJ 08544
(Voice) 609 258 4108, (Fax) 609 258 3745
Email: vivek@ee.princeton.edu

**Abstract**

Recent years have witnessed a rapid growth in research activity targeted at reducing energy consumption in microprocessor based systems. However, this research has by and large not recognized the potential energy savings achievable through optimization of software running on the microprocessor. This paper presents an overview of techniques used in our work and in other recent research in this area. Using the results of a recent work [9] as a basis, several possible techniques for energy reduction through code compilation are presented. Examples with energy reduction of up to 40% on an Intel 486DX2 based system, obtained by rewriting code, demonstrate the potential of these ideas. Several additional avenues for reducing CPU and memory system energy through code compilation are identified. The effect of traditional compilation techniques on energy reduction is discussed and some of these techniques that can be beneficial in this regard are reviewed.

## Introduction

The potential of energy reduction through modification of software has by and large been ignored by the low-power research community. This was largely due to the fact there was a lack of practical techniques to analyze the energy consumption of programs. This deficiency has been alleviated by a recent work [9]. That work presents an instruction-level model for energy consumption in processors, which quantifies and explains the energy variation from program to program. The growing trend towards tight energy budgets necessitates identification and exploration of every possible source of energy reduction, forcing us to examine the design of energy efficient software. This paper uses the results of the above work to study the issue of compilation for energy efficient code. Several avenues for energy reduction through compilation are presented in the subsequent sections.

## Reordering Instructions to Reduce Switching

The switching activity in a circuit is a function of the present inputs and the previous state of the circuit. Thus it is expected that the energy consumed during execution of a particular instruction will vary depending on what the previous instruction was. Thus an appropriate reordering of instructions in a program can result in lower energy. A recent work [5] presents a technique for scheduling instructions on an experimental RISC processor in such a way that the switching on the control path is minimized. The paper reports that a significant reduction in the switching activity is obtained through this method.

Our experiments based on actual energy measurements on the 486DX2 however reveal that the reduction in switching activity achieved by this technique does not translate into very significant overall energy reduction. This technique is trying to reduce what is termed as the *circuit-state overhead* [9], a quantity that is bounded by a small range and that does not show a great amount of variation. In fact, it was observed that different reorderings of several sequences of instructions showed a variation of only up to 2% in their energy cost. While this technique does not appear to be very effective for the 486DX2, it is targeting a real source of energy consumption and thus its effectiveness on other architectures should be verified, preferably through actual measurements.

## Code Generation through Pattern Matching

The instruction selection module of modern code generators is often generated automatically by programs called *code-generator-generators*. `IBURG` [4] is such a program that is used to generate code for `lcc`, a retargetable `ANSI C` compiler. These programs accept a specification of the instruction set of the targeted architecture. The specification defines the set of instruction patterns that can be included in the generated code, and assigns a cost for each pattern. A code generator is then generated. At compile time, this code generator accepts an intermediate representation (DAG) of each basic block in the source code and using pattern matching and dynamic programming, tries to find a cover for the DAG in terms of the specified instruction patterns in such a way that the overall cost is minimized.

The cost function used in most compilers, including `lcc`, is the number of execution cycles, but it can be changed to energy costs to obtain a code generator that targets energy consumption. This was done for `lcc` using the experimentally determined 486DX2 energy costs. Interestingly, it was observed that the energy based and cycle based code generators produced very similar code. The reason for this is that the energy cost of an instruction pattern equals average power times the number of clock cycles. Now the power and cycle costs of the specified instruction patterns were such that the minimum energy pattern also had the minimum number of cycles and there was limited power variation among patterns with same cycle cost. Nevertheless, this instruction selection technique is a powerful technique for choosing among alternative instruction sequences and its use in energy based compilation deserves further investigation, especially for other microprocessors.

## Reduction of Memory Operands

An inspection of the energy costs of 486DX2 instructions reveals that instructions with memory operands have very high energy costs compared to instructions with register operands. Instructions using only register operands cost in the vicinity of $300mA$ per cycle. Memory reads that hit the cache cost upwards of $430mA$. Memory writes cost upwards of $530mA$ and also incur a memory system energy cost since the cache is write-through. Potential pipeline stalls, misaligned accesses and cache misses, further add to the cost.

| Program | hlcc.asm | hht1.asm | hht2.asm | hht3.asm |
|---|---|---|---|---|
| Avg. Current ($mA$) | 525.7 | 534.2 | 507.6 | 486.6 |
| Execution Time ($\mu$sec) | 11.02 | 9.37 | 8.73 | 7.07 |
| Program | clcc.asm | cht1.asm | cht2.asm | cht3.asm |
| Avg. Current ($mA$) | 530.2 | 527.9 | 516.3 | 514.8 |
| Execution Time ($\mu$sec) | 7.18 | 5.88 | 5.08 | 4.93 |

Thus reduction in the number of memory operands can lead to large energy savings.

Reduction in number of memory operands can be achieved by adopting suitable compiler policies, e.g. saving the least amount of context during function calls. However, the most effective way of reducing memory operands is through better utilization of registers. This entails optimal register allocation of temporaries and global register allocation for the most frequently used variables. Use of register operands as opposed to memory operands reduces energy since it reduces average power. In fact, it reduces energy further since it can also lead to shorter running times due to elimination of potential stalls and cache misses. The reduction in running time due to the use of registers has led to a vast body of research dealing with the problem of register allocation and its interaction with the instruction ordering and selection phases of compilation [1] [2]. While this research has immediate application in compilation for low energy, some points should be noted in this regard.

First, compilers do not always choose the most aggressive of the known techniques. However, the energy savings obtainable from these techniques can more than compensate for the overhead of longer compilation time and compiler complexity, especially for commercial software. Second, presence of large on-chip caches with high hit rates and 1 cycle access time, may make register allocation less important from the point of view of performance but it is still very important from the point of view of energy, since the average power per cache access is much higher than accessing a register. And finally, traditional performance driven register allocation algorithms may need to be modified to achieve maximum energy savings - e.g. to account for the fact that due to the presence of a write-through cache, writing a variable has a different (higher) cost than reading it.

The next section illustrates through examples, the potential of the ideas presented in this section. Details of the ongoing work regarding the ideas in the above sections can be found in [8].

### Illustration of Software Energy Optimization

The first program considered is a *heapsort* program in C. `hlcc.asm` is the assembly code for this program generated by `lcc`. The sum of the observed average CPU and memory currents is given in the table above. The program execution times are also reported. `lcc` is a general purpose compiler and while it produces good code, it leaves room for further improvement of running time. Hand tuning of the code for shorter running time (`hht1`) leads to a 13.5% reduction in energy even though the average current went up. So far only temporary variables had been allocated to registers. In `hht2`, 3 local variables are allocated to registers and the appropriate memory operands are replaced by register operands. Even though redundant instructions are not removed, there is a 5% reduction in current and a 7% reduction in running time. In `hht3`, 2 more local variables are allocated to registers and all redundant instructions are removed. Compared to `hlcc`, `hht3` has 40.6% lower energy consumption. Significant energy savings are also observed for the other program shown in the table.

### Other Avenues for Energy Reduction

This section identifies the other possible sources of energy reduction that can be exploited during compilation. Pointers to existing work are provided. Even though these works may not consider energy explicitly, they are still relevant. Each of the following is a potential source of energy reduction, but the effectiveness may not be of the same magnitude as the methods described in the previous section. But as stated earlier, in the scenario of tight energy budgets, all sources of energy reduction should be explored.

- Scheduling to reduce pipeline stalls. These stalls take up cycles and consume energy.
- Reducing memory accesses by coalescing narrow memory references into wide ones [3].
- Code transformations to improve cache hit rates [7].
- Reducing switching on address lines [6].
- In page-mode DRAM chips, page misses consume more energy [9]. Thus methods to improve page hit ratio can be beneficial, e.g. reordering of memory accesses [7] and allocation, blocking and copying of data.

### Concluding Remarks

In this paper we have shown that the process of compilation can effect the energy consumption of the software running on a microprocessor. We have presented several techniques for optimizing this energy consumption. Compilers play an important role in exploiting the capabilities of an architecture. This is reflected in the fact that modifications in processor architecture for better code performance often need to consider how compilers will react to the modifications. It can be concluded from the work presented here that the same is true even in the case of energy consumption.

# References


[1] A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers, Principles, Techniques and Tools.* Addison Wesley, 1988.

[2] M. Benitez and J. Davidson. A retargetable integrated code improver. Technical Report CS-93-64, Univ. of Virginia, Dept of Computer Sc., Nov. 1993.

[3] J. Davidson and S. Jinturkar. Memory access coalescing: A technique for eliminating redundant accesses. In *Proceedings of PLDI*, June 1994.

[4] C. Fraser et al. Engineering efficient code generators using tree matching and dynamic programming. Technical Report CS-TR-386-92, Princeton Univ., Aug. 1992.

[5] C. L. Su et al. Low power architecture design and compilation techniques for high-performance processors. In *IEEE COMPCON*, Feb. 1994.

[6] S. Wuytack et al. Global comm. and memory optimizing transformations for low power systems. In *International Workshop on Low Power Design*, April 1994.

[7] S. A. McKee. Access ordering and memory-conscious cache utilization. Technical Report CS-94-10, Univ. of Virginia, Dept of Computer Sc., March 1994.

[8] V. Tiwari, S. Malik, and A. Wolfe. Compilation techniques for low power. In preparation.

[9] V. Tiwari, S. Malik, and A. Wolfe. Power analysis of embedded software: A first step towards software power minimization. Technical Report CE-M94-4, Princeton Univ., Dept. of Elect. Eng., April 1994.